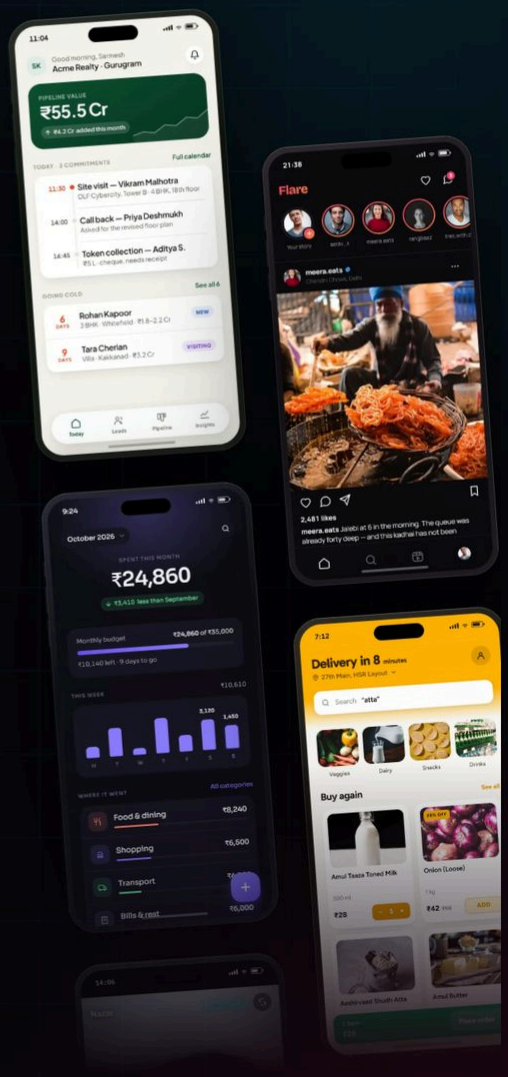
NEAT ROOTS

4 MONTHS · LIVE | HINGLISH | START FROM ZERO

FLUTTER FORMULA · THE MENTORSHIP PROGRAM

Become a Flutter developer who can actually ship.

4 months of live mentorship. You'll publish your own app on the Play Store, and build the same production CRM I ship to paying clients. **No prior coding experience needed.**



12 October	Mon · Wed · Fri	50 seats	5 apps
BATCH STARTS	8 TO 10 PM IST	CODE REVIEWED LIVE	ONE ON PLAY STORE

I'm **Sarmesh**. I teach Flutter to **300,000+ developers on YouTube**, and I build Flutter apps for paying clients. Everything in this program is what I actually do at work, not a syllabus I read somewhere.

THE POINT OF ALL THIS

Where you'll be in 4 months.

Job-ready

Walk into a Flutter interview knowing how production code is actually written.

Published

Your own app live on the Play Store, with your name on it.

A resume that stands out

A real multi-tenant CRM, not another to-do app clone.

Freelance-capable

Ready to take on Flutter projects of the kind that go for $1,000 to $2,000 in the market.

4 months	Live classes	3 per week	50 seats
DURATION	NOT RECORDINGS	6 HOURS OF CLASS	EVERY STUDENT REVIEWED

BEFORE YOU DECIDE

Is this program for you?

You're in your engineering or IT degree.

You've cleared DSA rounds on paper but you've never built something a stranger could install and use. Placements are coming and your resume has college projects that look like everyone else's. You need one project that makes an interviewer ask a follow-up question.

You're from a non-IT background.

Different degree, different job, or no job yet, and you've decided you want to build apps. You've watched a few videos and quit because everyone assumes you already know something. This program starts from what a variable is. Zero coding knowledge is genuinely fine here.

You've been "learning Flutter" for months.

You can follow a tutorial and the app works. Then you open a blank project on your own and nothing comes. The gap isn't your effort. It's that nobody showed you how real projects are actually built.

HONEST FILTER

This is not for you if

You can only watch recordings.

These are live classes. Recordings are there so you can revise, not so you can skip. Students who only watch recordings fall behind and stop coming.

You can't give 3 to 4 hours a day.

Class time is not enough. You practise, you build, you submit. If your schedule doesn't have that room, take the next batch instead of losing your money on this one.

You plan to catch up later.

The program builds forward every week. A one-week gap becomes a two-week backlog, and that is where people quit. Consistency matters more than talent.

What you actually need

A laptop

Windows, Mac or Linux. 8GB RAM is enough.

3 to 4 hrs a day

Including class time.

Zero coding

We start from what a variable is.

Show up live

And submit your work.

HOW IT'S STRUCTURED

Four phases. Each ends with something you couldn't do before.

Pace adjusts to the batch. Phases run in this order, but we slow down where students need it.

PHASE 01

01

Programming from zero

Dart from the very first line: variables, logic, loops, collections, functions, classes, async. Git and GitHub from week one. Small command line programs so the logic sticks before any UI shows up.

By the end: you can read and write Dart without copying from anywhere.

PHASE 02

02

Your first real Flutter app

Widgets, layouts, state, forms, navigation, theming, dark mode, responsiveness, animations, and debugging with DevTools. You build the Expense Tracker end to end.

By the end: you can build a complete Flutter app without a tutorial open in another tab.

PHASE 03

03

How production apps are built

Where most self-taught developers stop. Riverpod, Clean Architecture, Feature-First structure, REST APIs with Dio, JWT auth, interceptors, Firebase, AWS. You build the CRM.

By the end: you can structure a codebase a team could work on, and explain your decisions.

PHASE 04

04

Scale, publish and ship

Maps and location, infinite scroll and performance, on-device ML, then release builds, signing, Play Console and publishing. Quick Commerce, Social Media, and your own app on the Play Store.

By the end: your app is live, and you can take any project from empty folder to published.

• WHAT YOU'LL BUILD

Five apps. One of them is why most students join.

Each one is harder than the last, and each one exists because it teaches something the previous one couldn't.

★ THE FLAGSHIP PROJECT

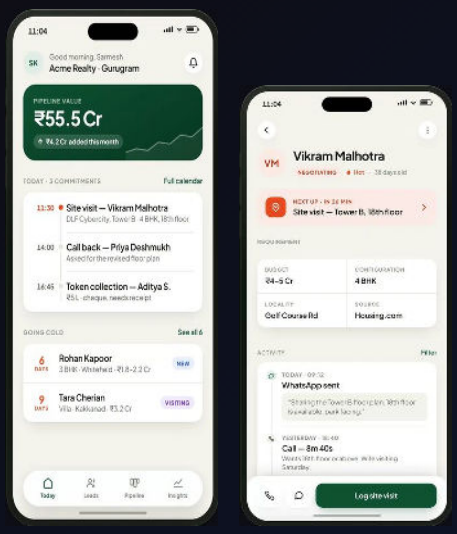
A business CRM, the same kind I ship to paying clients.

CRM is the most common thing businesses actually pay Flutter developers to build. You build it the way I build it at work: same architecture, same auth flow, same folder structure, same mistakes avoided.

Everyone builds the same core: authentication, customers, leads, follow-ups, task assignment. Then you pick a vertical and build the modules that business needs, together in class, module by module.

Real estate · Clinic · Salon · Coaching

Fifty students. Fifty different CRMs. Yours is the one you can talk about for twenty minutes in an interview.



• THE FLAGSHIP, UNPACKED

What's inside the CRM

THE CORE EVERYONE BUILDS

- Employee auth with JWT
- Customer management
- Lead pipeline
- Follow-up system
- Task assignment
- Image upload
- Search and filters
- Pagination
- Push notifications
- Offline support

YOUR VERTICAL LAYER

- The modules only your chosen business needs
- Built live in class, with you
- Real estate, clinic, salon or coaching institute

WHAT YOU'LL ACTUALLY LEARN

- REST APIs with Dio
- JWT auth and refresh tokens
- Clean Architecture
- Riverpod
- Repository pattern
- Firebase
- AWS S3
- A production folder structure you'll reuse on every project

• ALL REAL, ALL WORKING

The other four projects

01 FOUNDATIONS

Spendly: Expense Tracker

Every Flutter developer starts with fundamentals. Instead of small, disconnected demos, you build one complete app that teaches the foundation of Flutter development.

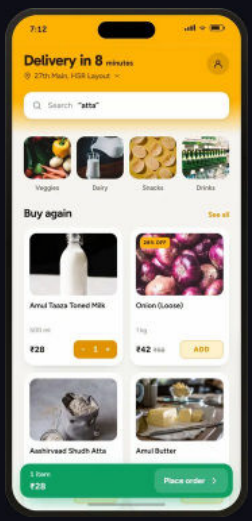
WHAT YOU'LL BUILD

- Modern UI
- Add, edit, delete expenses
- Categories
- Charts & analytics
- Local storage
- Dark mode
- Responsive layout
- Animations

SKILLS YOU'LL GAIN

- Widgets & layout
- Forms & validation
- Navigation
- Local database
- State management

OUTCOME You can build complete Flutter apps without relying on tutorials.



02 LOCATION-BASED

QuickKart: Quick Commerce (Blinkit style)

Location-based apps, and how modern delivery platforms work behind the scenes.

WHAT YOU'LL BUILD

- Product catalog & search
- Cart & checkout
- Delivery address selection
- Google Maps / Mapbox
- Current location
- Route visualisation
- Nearby store detection
- Push notifications
- Order history

SKILLS YOU'LL GAIN

- Maps integration
- Geolocation
- Reverse geocoding
- Location permissions
- API integration
- Business logic design

OUTCOME You understand how location-aware apps work, one of the most common app categories.

03 CAPSTONE

Flare: Social Media (Instagram style)

The capstone. Everything you have learned, combined into one production-scale application.

WHAT YOU'LL BUILD

- Auth & profiles
- Feed & posts
- Image upload
- Likes & comments
- Follow / unfollow
- Reels
- Infinite scroll
- Search users
- Saved posts

SKILLS YOU'LL GAIN

- Production architecture
- Media upload
- Infinite pagination
- Performance & caching
- Real-time features
- Advanced animations

OUTCOME A portfolio-worthy app that shows you can build production-ready mobile apps.

★ BONUS · AI / ML

FaceLens: Face Detection

On-device machine learning in Flutter with Google's ML Kit, no data science background needed.

WHAT YOU'LL BUILD

- Real-time detection via camera
- Detection from gallery
- Multiple faces
- Face landmarks
- Smile probability
- Eyes open / closed
- Face tracking
- Bounding box overlay

SKILLS YOU'LL GAIN

- google_mlkit_face_detection
- Camera frame streaming
- Image processing
- CustomPainter overlays
- On-device ML models

OUTCOME You can add AI features to any Flutter app and use ML Kit's other capabilities with confidence.

EPISODE BY EPISODE

The full curriculum

Ten episodes, from installing Flutter to shipping on the Play Store and working with AI.

01 EPISODE 01 · 10 TOPICS — Getting Started

- How this program works, and how to get the most out of it
- What programming actually is, for anyone starting from zero
- How a computer runs the code you write
- What a mobile app is actually made of
- What Flutter is, and why it exists
- Flutter vs React Native vs native Android and iOS
- Installing the Flutter SDK, Android Studio and VS Code
- Setting up an emulator, and running on your own phone
- Your first Flutter app running on screen
- Fixing the setup problems everyone hits on day one

02 EPISODE 02 · 45 TOPICS — Git and GitHub

Introduction
- What is Version Control?
- Why do developers need Git?
- Git vs GitHub
- Git vs Google Drive (analogy)

Installing Git
- Installation
- Git Bash
- Configure Username & Email

Git Basics
- Repository
- Initialize Repository (git init)
- Staging Area
- Working Directory
- Commit History

Basic Git Commands
- git init
- git status
- git add
- git commit
- git log
- git diff
- git restore
- git rm
- git mv

Branching
- Why Branches?
- Create Branch
- Switch Branch
- Merge Branch
- Delete Branch

Merge Conflicts
- Why conflicts happen
- Resolve conflicts
- Best practices

GitHub
- Create GitHub Account
- Create Repository
- Public vs Private
- Clone Repository
- Push
- Pull
- Fetch

.gitignore
- Why use it
- Flutter .gitignore
- Ignore build folder
- Ignore API keys
- Ignore .env

Git Best Practices
- Commit Message Convention
- Small Commits
- Feature Branch Workflow
- Never Commit Secrets

03

EPISODE 03 · 99 TOPICS

Dart Fundamentals

Introduction

- How to make the most of this Dart course
- Introduction to Dart
- Introduction to DartPad

Dart Basics

- The main method
- Hello World
- Variable declaration and initialization
- Basic types
- [Exercise] Printing variables
- String concatenation & interpolation
- [Exercise] String interpolation
- String escaping
- Multi-line strings
- Basic String operations: uppercase and lowercase
- Initialization vs Assignment
- Finding and replacing strings
- Conversions between types
- Arithmetic operations
- [Exercise] Temperature conversion
- Increment & decrement operators
- Logical & Relational operators
- Ternary operator
- Hex format, bitwise & shifting operators
- Comments
- Expressions & statements

Dart Type System

- Static vs Dynamic Languages
- Type inference with var
- The final keyword
- The const keyword
- [Exercise] var, final and const
- The dynamic keyword

Control Flow

- if-else statements
- while loops
- for loops
- [Exercise] Fizz buzz
- Break and Continue
- Switch Statements
- Enumerations
- [Exercise] Simple Calculator

[Project] Build a Command Line App

- Installing the Dart SDK
- Installing and configuring VS Code
- Project brief: Rock, Paper & Scissors
- Creating a command line app
- Pseudocode for the game logic
- Getting user input with stdin from dart:io
- Implementing the game loop

Collections

- Lists
- [Exercise] Sum of the items in a list
- List methods
- Type annotations with lists
- Using var, final, const with lists
- Sets
- [Exercise] Sets
- Maps
- The as operator
- Null values
- Iterating on maps
- [Exercise] Pizza Ordering
- Nested Collections
- [Exercise] Restaurant ratings
- Collection-if
- Collection-for
- Spreads
- [Exercise] Shopping List
- Copying collections

[Project] Data Processing in Dart

- Parsing command line arguments
- Reading files line by line
- Pseudocode for the processing logic
- Implementing the processing logic

Null Safety

- Introduction to Null Safety
- Nullable and non-nullable variables
- Flow Analysis, Promotion and Definite Assignment
- The assertion operator
- The if-null operator
- Null Safety with type inference
- Null Safety with collections
- The conditional access operator & the billion dollar mistake

Functions Basics

- Intro to functions
- Function arguments
- Return values
- [Exercise] Sum of a list of numbers
- Named and positional arguments
- Required and default values
- Default positional arguments
- [Exercise] Pizza ordering with functions
- Fat arrow notation (=>)
- The global and local scope
- Inner Functions
- Global mutable state and functions with side effects

Functions Advanced

- Anonymous functions
- Functions as first class objects
- Function types
- Closures
- The forEach method
- The map method
- Iterable and toList()
- Code reuse with anonymous functions and generics
- The where and firstWhere methods
- The reduce method
- Combining functional operators

04 EPISODE 04 · 55 TOPICS

Object-Oriented Dart

Classes Basics

- Introduction to classes
- Instance methods
- Initializer
- Classes with immutable members
- [Exercise] Creating a Person class
- Type safety with classes
- const constructors
- Named constructors
- Named constructors: temperature example
- Getters and setters
- [Exercise] Restaurant ratings with classes
- Static methods and variables
- Private variables and methods

Classes Advanced

- Introduction to inheritance
- The super constructor
- Overriding methods
- Abstract classes
- [Exercise] Area and Perimeter
- Interfaces: implements vs extends
- The base Object class
- The toString() method
- The equality operator and the covariant keyword
- [Exercise] Implement the + and - operators
- Overriding hashCode and the Equatable package
- Using classes with generics
- Composition vs inheritance: Flutter widget hierarchy
- Factory constructors and reading JSON data
- [Exercise] JSON Serialization
- Copying objects with copyWith
- The cascade operator

[Project] Simple eCommerce App

- Overview
- Creating the Product, Item, Cart classes
- Adding the interactive prompt
- Adding items to the cart
- Checkout functionality
- Project structure and wrap-up

Mixins & Extensions

- Creating and using mixins
- Mixins drawbacks
- Extensions
- Extensions with generic type constraints
- [Exercise] Range extension

Error Handling & Exceptions

- Errors vs Exceptions
- Assertions
- Exceptions: throw, try, catch, finally, rethrow
- [Exercise] Email validation

Asynchronous Programming

- Futures: then, catchError, whenComplete
- async and await
- Future.value and Future.error
- [Exercise] Countdown with Futures
- Streams
- Stream generators: async* and yield
- [Exercise] Fizz-buzz with streams
- Stream constructors
- Stream methods
- Single vs multiple subscription streams

05 EPISODE 05 · 90 TOPICS

Flutter Fundamentals

Introduction to Flutter

- What is Flutter
- Project creation & setting up a code editor
- Running a first Flutter app
- Understanding Material Design
- Analyzing a new Flutter project
- From Dart to machine code
- How the programming language works in Flutter
- Importing features from packages
- How Flutter apps start

Widgets I: A Solid Foundation

- Understanding widgets
- Using a first widget & passing values to functions
- Combining multiple widgets
- Understanding const values
- Building more complex widget trees
- Configuring widgets & understanding objects
- Generics, lists & gradient colors
- Practice: styling text
- Why you need custom widgets
- Building custom widgets
- Instance variables & configurable widgets
- Practice: reusable widgets & constructors
- Displaying images & multiple constructors
- Adding buttons & functions as values
- Styling buttons & working with padding
- How NOT to build interactive widgets
- Introducing Stateful widgets
- Generating random numbers

Widgets II: Fundamentals Deep Dive

- A challenge for you!
- Adding icons to buttons

- Adding transparency to widgets
- Rendering content conditionally
- Accepting & passing functions as values
- The initState method
- Ternary expressions & comparison operators
- Understanding if statements
- Adding a data model & dummy data
- Configuring a Column
- Creating a reusable, custom styled button
- Alignment, margin & padding
- Mutating values in memory
- More on button styling
- Third-party packages & Google Fonts
- Passing data via functions across widgets
- Combining Columns & Rows
- Expanded to the rescue!
- Scrollable content with SingleChildScrollView

Navigation & Routing

- Why an app needs more than one screen
- Navigator, push and pop
- Passing data between screens
- Getting data back from a screen
- Named routes, and where they stop working
- Introduction to go_router
- Declarative routes and route paths
- Route parameters
- Nested navigation with a bottom navigation bar
- Route guards: keeping screens behind login
- Handling unknown routes

Debugging Flutter Apps

- The starting project & a problem
- Understanding error messages
- Debugging apps & using debug mode
- Working with the Flutter DevTools

Interactivity, More Widgets & Theming

- AppBar with a title & actions
- User input with TextField
- Getting user input on every keystroke
- TextEditingController
- Adding a Dropdown button
- Validating input & showing an error dialog
- Dismissible list items
- Showing & managing Snackbars
- Getting started with theming
- Setting & using a Color Scheme
- Setting Text Themes
- Using theme data in widgets
- Adding Dark Mode
- Adding chart widgets

Responsive & Adaptive UI [Expense Tracker]

- What is responsiveness
- Locking the device orientation
- Updating the UI based on available space
- Understanding size constraints
- Handling screen overlays like the soft keyboard
- Understanding Safe Areas
- Using the LayoutBuilder widget
- Building adaptive widgets

Adding Animation

- Explicit vs implicit animations
- Explicit: adding an AnimationController
- Explicit: playing it with AnimatedBuilder
- Fine-tuning explicit animations
- Getting started with implicit animations
- Configuring implicit animations
- Adding multi-screen animation

06 EPISODE 06 · 34 TOPICS

State Management

Why State Management?

- What "state" actually means in Flutter
- setState and where it stops working: prop drilling, rebuilding too much, no sharing across screens
- The exact wall you hit in the Expense Tracker
- What a state management package really solves, and what it does not
- How professionals choose: by the shape of the state (local, shared, async)

Provider: The Foundation

- Why Provider first: you will meet it in almost every existing codebase
- InheritedWidget: what Provider is built on
- Installing provider
- ChangeNotifier & ChangeNotifierProvider
- Consumer, context.watch, context.read, context.select
- MultiProvider
- ProxyProvider: providers that depend on other providers
- Combining local widget state with Provider state
- Where Provider falls short

Riverpod: What We Actually Build With

- Why Riverpod: same author as Provider, written to fix its holes
- Installing flutter_riverpod & ProviderScope
- Provider, StateProvider, NotifierProvider, AsyncNotifierProvider
- ref.watch, ref.read, ref.listen, and when to use which
- FutureProvider: loading API data without a StatefulWidget
- Composing providers
- autoDispose & family
- Where providers live in a Feature-First structure
- Wiring Riverpod into the CRM: auth state, leads list, follow-ups

Industry Landscape: GetX & BLoC

- Why this module exists: freelance and agency codebases are full of these
- GetX: GetxController, Obx, Get.put, Get.find, GetMaterialApp
- Reading and maintaining an existing GetX codebase
- Why we do not build with GetX here
- BLoC: Events, States, Bloc vs Cubit
- Recognising BLoC in an existing project
- Working inside someone else's state management without a rewrite

Choosing State Management in Real Projects

- The decision rule: local & short-lived: setState. Shared or async: Riverpod
- Migrating a screen from Provider to Riverpod
- Common mistakes: everything global, rebuilding the whole screen, logic inside widgets
- Practice: refactor the Expense Tracker from setState to Riverpod

07 EPISODE 07 · 19 TOPICS

Clean Architecture

Introduction

- Why Clean Architecture matters: scalability, maintainability, team workflows
- Common problems without architecture: tight coupling, one change breaking five files
- Where Clean Architecture fits in Dart and Flutter

Architecture Layers

- Entities (Domain Models): pure business rules, framework-independent
- Use Cases (Application Layer): one use case, one responsibility
- Interface Adapters: repository implementations, DTOs, mappers
- Frameworks & Drivers: UI, API clients, database, Flutter widgets

Feature-First Approach

- Layer-First vs Feature-First
- Why Layer-First breaks at scale
- A feature is a vertical slice: data + domain + presentation
- Folder structure: core/, features/ (data, domain, presentation), routing/, main.dart
- Rules: one feature, one self-contained folder
- A feature never imports another feature's internals
- The deletion test: delete a feature, the app still compiles
- Dependency direction: presentation, domain, data
- Mapping CRM modules to features: auth, customers, leads, follow-ups, tasks
- Common mistakes: core/ as a dumping ground, over-engineering
- Practice: refactor the Expense Tracker to Feature-First
- Practice: add a CRM feature without editing any existing one

08

EPISODE 08 · 51 TOPICS

Backend, APIs and Cloud

Connecting a Backend & HTTP Requests

- What a backend is and why you want one
- What HTTP is & how it works
- Adding the Dio package
- Sending a POST request
- Working with the request & waiting for the response
- Fetching & transforming data
- Avoiding unnecessary requests
- Managing the loading state
- Error response handling
- Sending DELETE requests
- Handling the no data case
- Better error handling
- Using the FutureBuilder widget

Dio Deep Dive

- Why Dio over the http package
- Base Options
- Logging
- Authentication headers
- Refresh token flow
- Request cancellation

Authentication

- JWT authentication
- Login flow
- Refresh token
- Secure storage
- Auto login
- Logout

Interceptors

- What interceptors are, and why real apps need them
- Problems they solve: duplication, scattered logic
- Where they fit: the network layer, never UI or Domain
- Request interceptor: headers, logging
- Response interceptor: validation, central success handling
- Error interceptor: 401, 403, 500, SocketException, Timeout

Firebase

- What Firebase is, and why developers use it
- Firebase vs a traditional backend
- Authentication: Email & Password, Google Sign-In
- Cloud Firestore: NoSQL, collections & documents
- Realtime Database: when to use it
- Firebase Storage: image & file uploads
- Cloud Messaging (FCM): push notification basics
- Analytics & Crashlytics

AWS (Amazon Web Services)

- AWS features for mobile app developers
- Upload images using an S3 bucket
- CloudFront

Offline First

- SharedPreferences
- Hive
- Cache strategy

Security

- API key protection
- Secure storage
- Environment variables
- Root / jailbreak detection (overview)

Notifications

- Local notifications
- Handling foreground and background notifications

09

EPISODE 09 · 45 TOPICS

App Publishing (Play Store / App Store)

Introduction

- Why publish an app?
- Development vs production build
- APK vs AAB
- Internal testing vs production

Preparing Your Flutter App

- Change app name
- Change package name
- Change app icon
- Splash screen
- Version name & version code

Release Build

- Debug vs release
- Generate release APK
- Generate AAB
- Verify release build

App Signing

- Why signing is required
- Generate keystore
- Store password & alias
- Configure key.properties
- Configure Gradle
- Verify signing

Google Play Console

- Create developer account
- Create new app
- Fill app details
- Dashboard overview

Store Listing

- App title
- Short description
- Full description
- Feature graphic
- Screenshots
- High resolution icon
- Category
- Contact information

Privacy & Compliance

- Privacy policy
- Data safety form
- Permissions
- Ads declaration
- Target audience
- Content rating

Uploading the App

- Upload AAB
- Create release
- Release notes

- Version management

Publishing

- Review changes
- Submit for review
- Google review process
- Common reasons for rejection

10 EPISODE 10 · 24 TOPICS

Working with AI as a Developer

AI in a Developer's Workflow

- Why AI is now part of the Flutter developer's toolkit
- Where AI actually helps vs where it hurts
- AI as a pair programmer, not a replacement

Choosing the Right Tool

- Cursor
- Claude Code
- GitHub Copilot
- Windsurf
- When to use which

Prompting for Flutter Tasks

- Prompts that produce production-quality output
- Feeding project context effectively
- Prompt mistakes that produce broken code
- Prompts for widgets, state, API layers, testing

AI for Everyday Development

- Generating models, DTOs, mappers, boilerplate
- Refactoring legacy code
- Writing tests
- Debugging errors and stack traces
- Understanding unfamiliar codebases quickly

- Documentation and code comments

AI + Clean Architecture

- Keeping AI code aligned with Feature-First
- Reviewing AI output before committing
- When AI-generated code breaks the architecture

Shipping Faster Without Losing Quality

- Building a personal AI workflow
- Code review checklist for AI-generated code
- Avoiding over-reliance

WHAT IT COSTS

Everything included. One price.

EARLY BIRD

₹2,999 ~~₹7,000~~ 57% Off

Until **7 October, 11:59 PM IST.** ₹3,999 after that.

FOR THE FULL FOUR MONTHS · EVERYTHING INCLUDED

Full refund within the first 3 classes

Sit in, see how I teach, and decide before class 4. After that, no refund.

Book your seat at neatroots.com

WHAT'S INCLUDED

- All live classes. 3 per week, for 4 months.
- A doubt session after every class, until the doubts are done.
- Recording of every class, so you can revise.
- Assignments, and your code reviewed.
- The full CRM build, including your own vertical.
- Nothing locked behind a higher tier.

WHY IT'S THIS LOW

Programs like this usually go for ₹25,000 to ₹40,000, and the ones charging that are not always giving more. I'd rather have fifty students who actually show up than five who could afford a premium price.

> "₹30,000 isn't a decision you get to make. Under ₹4,000 is. YouTube already pays my bills. This program doesn't have to."
>
> Sarmesh, on why it costs this little

WHAT YOU'LL ACHIEVE

By the end, you'll have built

- ✓ 4 complete Flutter apps, plus a bonus AI project
- ✓ Business & enterprise software
- ✓ A location-based mobile app
- ✓ A social media platform
- ✓ Production-ready architecture
- ✓ Backend integration
- ✓ Firebase & AWS integration
- ✓ Maps & geolocation features
- ✓ Authentication systems
- ✓ Play Store ready applications

More importantly, you'll learn **how to think like a Flutter engineer**: breaking down complex problems, designing scalable architectures, and shipping apps that reflect real-world practice rather than tutorial examples.

QUESTIONS

Questions people actually ask

Isn't all of this free on YouTube?

A lot of it is, and I put it there myself. What YouTube can't give you is someone watching your code and telling you what's wrong with it. It can't tell you why your architecture will break in month three. It can't keep you going in week six when you want to quit. **Free content teaches topics. This teaches you to build, and it keeps you accountable while you do.**

When are the classes?

Monday, Wednesday and Friday, 8:00 PM to 10:00 PM IST. Every class ends with a doubt session that runs until the doubts are done.

What if I miss a class?

Every class is recorded and you get access to it. But recordings are for revision, not for replacing the class. Students who switch to recordings-only are the ones who fall behind and stop showing up.

How much time do I need outside class?

Class is 6 hours a week. Plan for **3 to 4 hours a day** in total, including class time. The building happens between classes, not during them.

I have zero coding experience. Will I keep up?

Yes. We start from what a variable is. Episode 1 covers what programming even is and how a computer runs your code, before Flutter shows up at all. You need time and consistency, not a background.

Will my laptop run this?

Windows, Mac or Linux, with 8GB RAM. That's enough.

Is there a job guarantee?

No, and be careful with anyone who promises one. What you get is a real project, production-level skills, and the ability to answer interview questions from experience instead of from memory.

What about internships?

Top performers get an internship at NeatRoots. It's a selection process based on your work through the program, not a guarantee for everyone. How many we take depends on the batch, and we'll tell you at the end of the course.

Can I get a refund?

Yes, within the first 3 classes. After that, no. Sit in, see how I teach, and decide before class 4.

How do I ask doubts?

At the end of every class. That's the session, and it isn't rushed.

Are the classes in Hindi or English?

Hinglish, the way developers actually talk. Slides and code are in English so you get used to reading real documentation.

NEAT ROOTS

Stop following tutorials.
Start shipping apps.

Four months, fully live, five real apps and one of them published under your own name.

BATCH STARTS 12 OCTOBER | 50 SEATS | ₹2,999 EARLY BIRD TILL 7 OCT

Enroll at neatroots.com

Scan or visit neatroots.com/flutter-mentorship-program